

DELPHI 7

Por Armando Dark_FirefoX [aht1980ster@hotmail.com]

Este curso es FreeWare, y esta acogido a la Licencia GNU - GPL. Su distribución es gratis, pero siempre con el debido espacio del nombre del autor.

1. ¿Por que he decidido crear este curso?

Bien, hace ya unos años me adentre en el mundo de la programación, no sabia por donde empezar, no sabia si aprender C, C++, Visual Basic, C#, Delphi. Finalmente, me decidí por Delphi, siendo un lenguaje de programación Orientado a Objetos (POO u OOP [*Object Oriented Programming*]). Además de esto Delphi se ha ido difundiendo por la Red; por lo que diferentes usuarios y compañías han ido creando mas componentes (***se explicará más adelante***) para el Delphi.

Siempre he estado buscando en Internet un curso que me enseñe y me ayude a programar en Delphi, siempre he encontrados Guías y Tutoriales muy antiguos. Con información que para versiones modernas del Delphi se pueden utilizar de una manera más fácil.

2. Introducción

¿Qué es Delphi?

Delphi es un lenguaje de programación de alto nivel, que soporta diseño de estructura y orientación a objetos. Se basa en el lenguaje de programación ObjectPascal, entre sus beneficios están la facilidad de leer el código sin perderse, rápida Compilación, y el uso de Múltiples **Units** para programación Modular.

3. Comenzando ...

Importante sobre Windows.

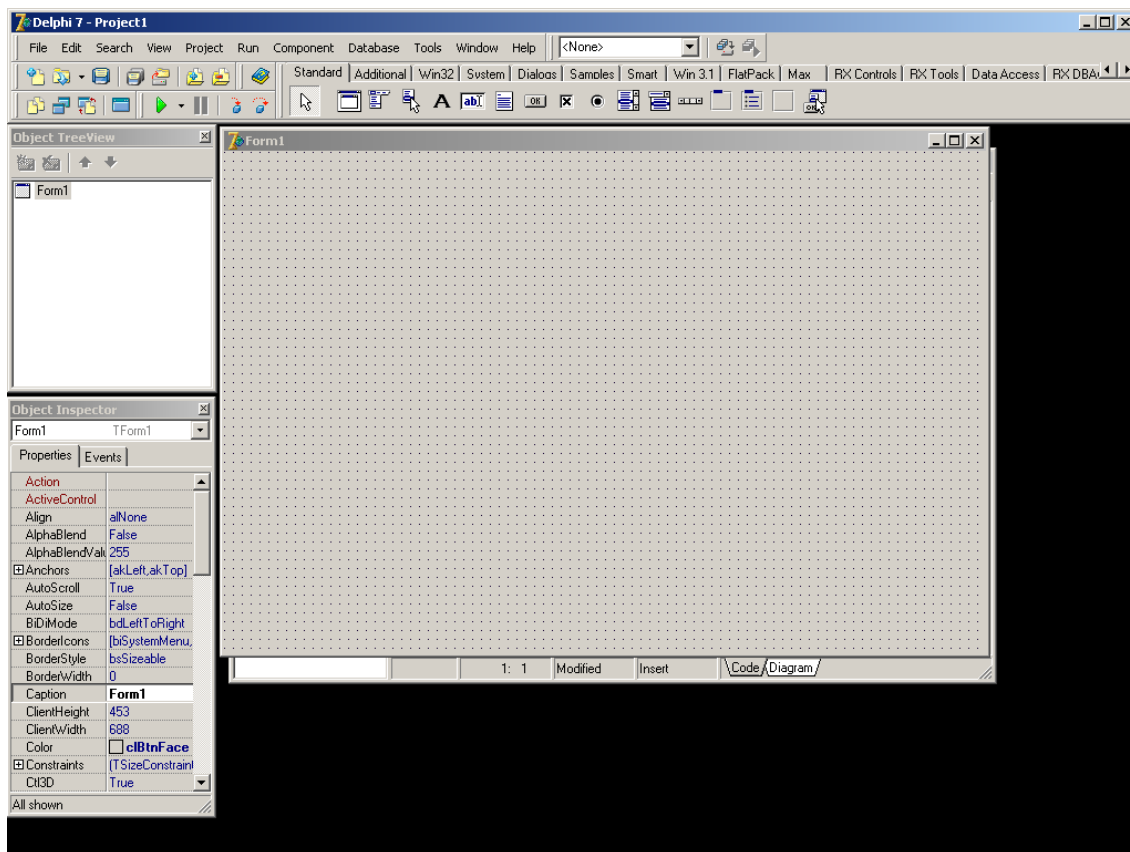
Bien, primeramente me gustaría aclarar que esta guía esta hecha para usuarios que tengan conocimiento del trabajo y manejo del Sistema Operativo Windows. Supongo que Ud. Conozca como manejar ventanas, usar programas, etc. En fin que sepa Trabajar en Windows como un usuario normal.

Funcionamiento de Windows.

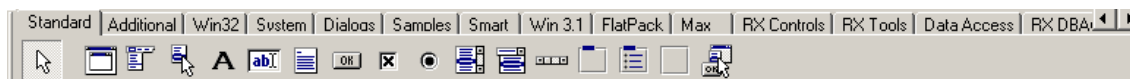
Debemos saber que Windows gestiona sus recursos (Mouse, Teclado, Pantalla) a través de mensajes, es decir cuando se mueve el ratón Windows envía un mensaje que diga que se movió el ratón, entonces es tarea de la aplicación en curso que hacer con el movimiento del ratón; la aplicación puede ejecutar un código, cambiar una propiedad, etc.; eso depende con fin que se haya realizado la aplicación.

Principios de la Teoría del Delphi

Visualización del Delphi



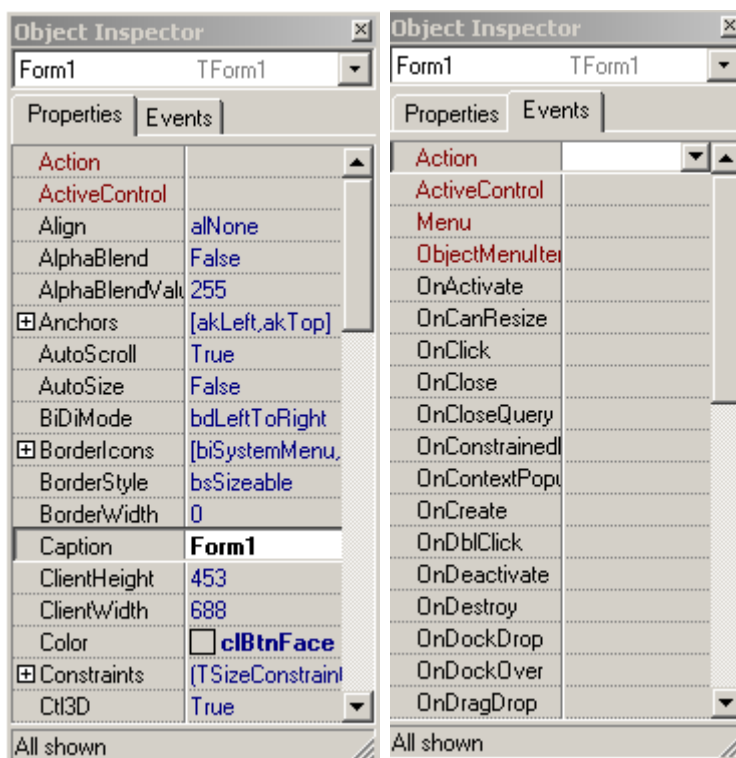
VCL (Visual Component Library o Librería de Componentes Visuales).



Delphi utiliza una librería de componentes llamada VCL (Visual Components Library) que nos brinda una serie de objetos y que nos da la posibilidad de utilizar los objetos sin necesidad de llamar a las distintas API de Windows para que dibujen en pantalla el componente, podremos además manipular a nuestro antojo el componente, cambiándole las propiedades como el nombre, el Caption o cosas así..., además de poder controlar de una manera muy fácil los distintos eventos que podría generar. A la hora de trabajar con los componentes mediante código, la sintaxis es la siguiente:

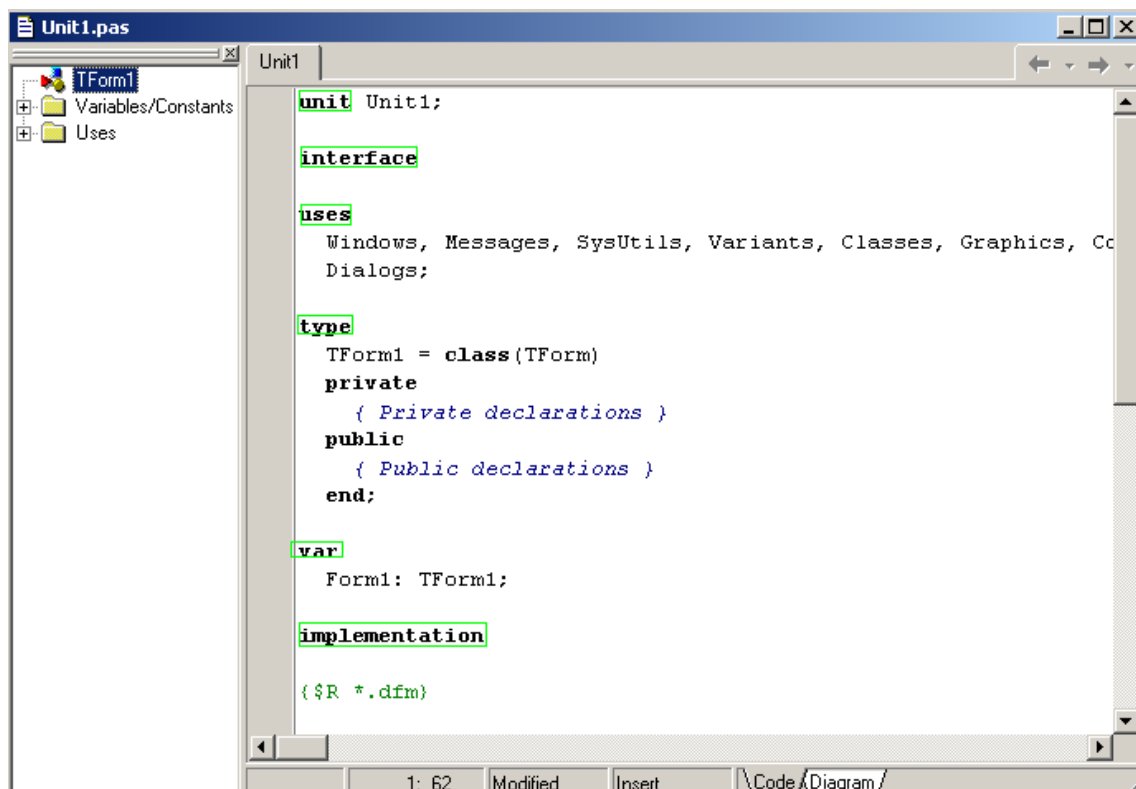
NombreDeComponente.PropiedadATrabajar

Inspector de Objetos (Object Inspector)



Se trata del Inspector de Objetos, gracias a él podemos cambiar de una manera rápida fácil y visual las propiedades de los distintos objetos que vayamos insertando en el programa. Además y gracias a él podemos controlar de forma fácil los distintos eventos, para ver los posibles eventos de un objeto selecciona la pestaña "**Events**", y para poder modificar el evento haz doble clic sobre el, ahora aparecerás en una especie de editor de textos llamada **Explorador de Code (Code Explorer)**

Explorador de Código (Code Explorer)



Veamos que en esta imagen hemos señalado unas palabras en un cuadrado verde, vamos a explicar que son estas palabras.

Estas palabras se llaman Palabras Reservadas, es decir que son palabras que Delphi las reconoce como parte de lo que el utiliza para reconocer el código. A continuación explicaremos las Palabras Reservadas puestas en la Imagen:

Unit : Hace referencia al nombre del documento de códigos.

Interface : Hace constar que lo que le sigue a esta palabra es parte de la interfaz del formulario.

Uses : En ella especificamos los distintos módulos que cargara nuestro programa y que se incluirán en el ejecutable una vez compilado para que esté tenga una independencia total del compilador, cuando insertamos un componente se añade a ella la clave para que se cargue el modulo que permite acceder a ese componente, así por ejemplo si añadimos un componente de la pestaña "Estándar" al ejecutar el programa se añade automáticamente la palabra "StdCtrls".

Type : En esta sección se especifican los distintos objetos con su correspondiente clase (NombreComponente:TipoComponente), Ej:(Label1:TLabel), y los distintos procedimientos que se ejecutaran en nuestro programa. Al final de ella veras 2 palabras clave "Private" y "Public", que sirven para definir variables privadas o publicas.

Var : Sirve para definir variables en cualquier sitio, si se pone en ese lugar, definiremos una variable que puede ser utilizada en todos los procedimientos del programa, si la definimos dentro de un procedimiento solo se podrá usar en él.

Implementation : Hace constar que el código a continuación forma parte ya de la codificación del programa, es decir el código que va a utilizar el programa para hacer funcionar a los componentes, en fin, para el funcionamiento para el que se va a crear

3.3.4.1 Operadores Aritméticos

Operador:	Delphi	C++	Visual Basic
Menor que...	a < b	a < b	a < b
Mayor que...	a > b	a > b	a > b
Igual que...	a = b	a = b	a = b
Desigual...	a <> b	a != b	a <> b
Menor o igual...	a <= b	a <= b	a <= b
Mayor o igual...	a >= b	a >= b+	a >= b
Asignación...	a := b	a == b	a = b
Comparación...	a = b	a = b	a = b
Division...	a / b	a / b	a / b
Multiplicación...	a * b	a * b	a * b

3.3.4.2 Comentarios

En delphi como en cualquier lenguaje de programación existe los comentarios, estos proporcionan la posibilidad de añadir a las líneas del programa, pequeñas anotaciones sobre como va el diseño del programa, por ejemplo podemos poner en una línea un comentario diciendo que es lo que sigue fallando para que al retomar el trabajo al día siguiente sepamos por donde empezar, en delphi existen 2 tipos principales de comentarios. Y son: Los que empiezan por " //" sirven para comentar una línea y los que están delimitados así: " {} " Sirven para poner un párrafo de comentario.

4.0 Comenzado a Programar

4.1 Variables

Una pieza clave en todas las aplicaciones son las variables y el buen control de estas. Una variable puede contener varios tipos de datos y pueden ser modificadas en todo el programa si las declara como globales, o bien por todos los formularios (Si las declara publicas) o en un determinado evento (Si las declara en ese evento).

Para definir una variable pública has de remitirte a la sección "Public" que encontraras debajo de "Type" y utilizar la siguiente sintaxis:

Variable : Tipodevariable;

Si lo que quiere es definir una constante que no vaya a variar en todo el programa utilice esta sintaxis:

Const Variable: Tipo = Valor;

Pero si lo que quiere es definir variables que puedan ser modificadas a lo largo del programa, deberá saber que en delphi existen varios tipos de variables, las más importantes son:

String	Cadena de texto
Integer	Numero
Byte	Byte de un archivo
Boolean	Afirmación o Negación (True or False)
Date	Una Fecha
Time	Un Tiempo
Char	Carácter.
Variante	La variable comodín, puede almacenar todo tipo de datos.

4.2 Convertir Variables

En delphi al contrario que en Visual Basic, necesitaremos definir las variables y además utilizarlas solo con los datos apropiados, así por ejemplo una variable tipo integer (Nº Entero) no podemos situarla en el texto de un edit, aunque este edit solo contenga números, para permitir esto, deberemos convertir el contenido de la variable a String; es decir que Delphi reconozca que el Numero que va a insertarse en el Edit sea tipo String.

A continuación les pongo una tabla con las conversiones mas utilizadas...

IntToStr	Convierte Integer a Texto
StrToInt	Convierte Texto a Integer
IntToHex	Convierte un Entero a un Hexadecimal
StrToDate	Convierte Texto en una Fecha
StrToTime	Convierte Texto en una Hora
DateToStr	Convierte Fecha en Texto
TimeToStr	Convierte Hora en Texto

4.3 Propiedades de los Objetos

Es importante conocer tanto los eventos propios de cada objeto como sus propiedades, por eso aquí les voy a mostrar una serie de propiedades generales que creo se adaptan a todos los objetos posibles.

Caption	Especifica el titulo del objeto o el texto que contendrá
---------	--

Cursor	Especifica el cursor que se mostrara cuando el ratón este sobre ese objeto
Default (Botones)	Especifica si ese botón Será el que se ejecute solo con dar un enter o no.
Enabled	Especifica si el objeto estará accesible al usuario
Font	Pos eso la fuente del texto de ese objeto
Height	El alto del objeto
Width	El ancho del objeto
Hint	El texto de explicación que mostrara cuando el ratón este sobre él
ShowHint	¿Mostramos o no la hint?
Visible	Especifica si Será visible o no.

4.4 Eventos

Un evento es la acción que se desencadena tras por ejemplo pulsar un botón, los eventos están controlados por los mensajes que emite Windows, así por ejemplo cuando pulsamos un botón, se produce un mensaje, el cual es recibido e interpretado por nuestro programa que hace que se desencadene el evento al que hace referencia ese mensaje.

Algunos de los eventos mas comunes y presentes en casi todos los objetos son:

OnCreate	Es el que se produce al crear el objeto
OnCloseQuery	Es el que se produce al cerrar la aplicación
OnDestroy	Se produce al destruir el objeto
OnKeyPress	Se produce al presionar una tecla sobre el objeto en el que estemos
OnChange	Se produce al cambiar alguna cosa del objeto (propiedades, contenido....)
OnClick	Se produce al hacer click sobre el objeto.
OnEnter	Se produce al situar el foco en un objeto
OnExit	Se produce cuando el objeto pierde el foco

4.5 Procedimientos (Procedure) y Funciones (Function)

Los procedimientos y las funciones son rutinas que se encargan de ejecutar una determinada acción, así por ejemplo podemos crear un procedimiento que se encargue de elevar a mayúsculas un texto, y luego llamarle para que convierta el texto que queramos.

```
Procedure ElevaMayusculas(Texto:String);
begin
  Uppercase(texto);
end;
```

```
{Y para llamarla en cualquier parte del programa haremos algo así}
ElevaMayusculas ('minúsculas');    // Elevamos a Mayusculas el texto
minúsculas'.
```

Este tipo de estructuras es útil para realizar determinadas acciones que se ejecutaran una y otra vez a lo largo del programa.

4.6 Condiciones

En casi todos los programas se utilizan condiciones, es decir:

"**Si** algo se cumple **entonces** haga algo, **sino** otra cosa"
Su sintaxis es realmente sencilla (casi intuitiva):

Lo explicado anteriormente se puede codificar de la siguiente manera:

```
If Condición=Verdadera then  
CODIGO  
Else  
CODIGO
```

Otra forma de expresar las condiciones es: ' **case of** ', lo que hace es seleccionar un identificador y ver que valores toma y en relación a ello optar por hacer algo o hacer otra cosa, este tipo de condición no acepta como identificador ni como valores a cadenas de texto, su sintaxis es la siguiente:

Lo explicado anteriormente se puede codificar de la siguiente manera:

```
case Identificador of  
1 : begin Showmessage ('Hola'); end; // En caso de que tome el valor 1,  
muestra un mensaje diciendo hola.  
2 : begin Showmessage ('Adios'); end; // En caso de que tome el valor 2,  
muestra un mensaje diciendo adios.  
end; // Acabamos
```

4.7 Bucles o Ciclos

Un bucle sirve para decirle al programa que ejecute cierta operación mientras una condición se cumple.

FOR

```
For I:= 0 to 100 do // Desde el contador (i) igual a 0 hasta 100 hacer (si  
fuera downto iría para abajo)  
begin  
CODIGO; // El código que se ejecutara...  
End; // Acabamos el bucle.
```

WHILE

```
While Condición do // Mientras se cumpla la condición...  
begin  
CODIGO; // El código que se ejecutará...  
End; // Acabamos el bucle.
```

REPEAT-UNTIL

```
Repeat // Repite...
```



```
V := v + 1; // Este codigo  
Until      // Hasta  
v > 100;    // Que v > 100
```

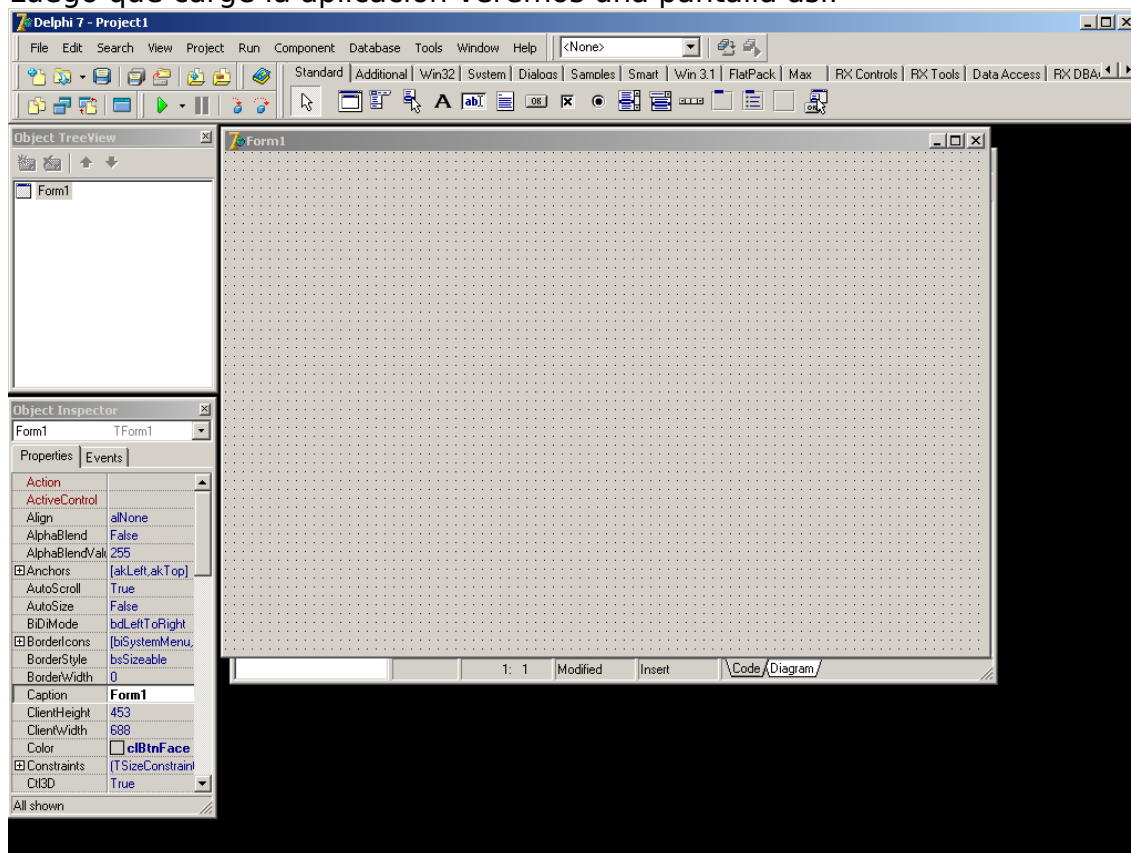
5.0 Primer Programa

5.1 Hola Mundo

Bien, casi siempre que leemos una guía de programación, nos damos cuenta que siempre se empieza con un programa llamado **Hola Mundo**, que lo único que hace es mostrar un mensaje que diga **“Hola Mundo”**.

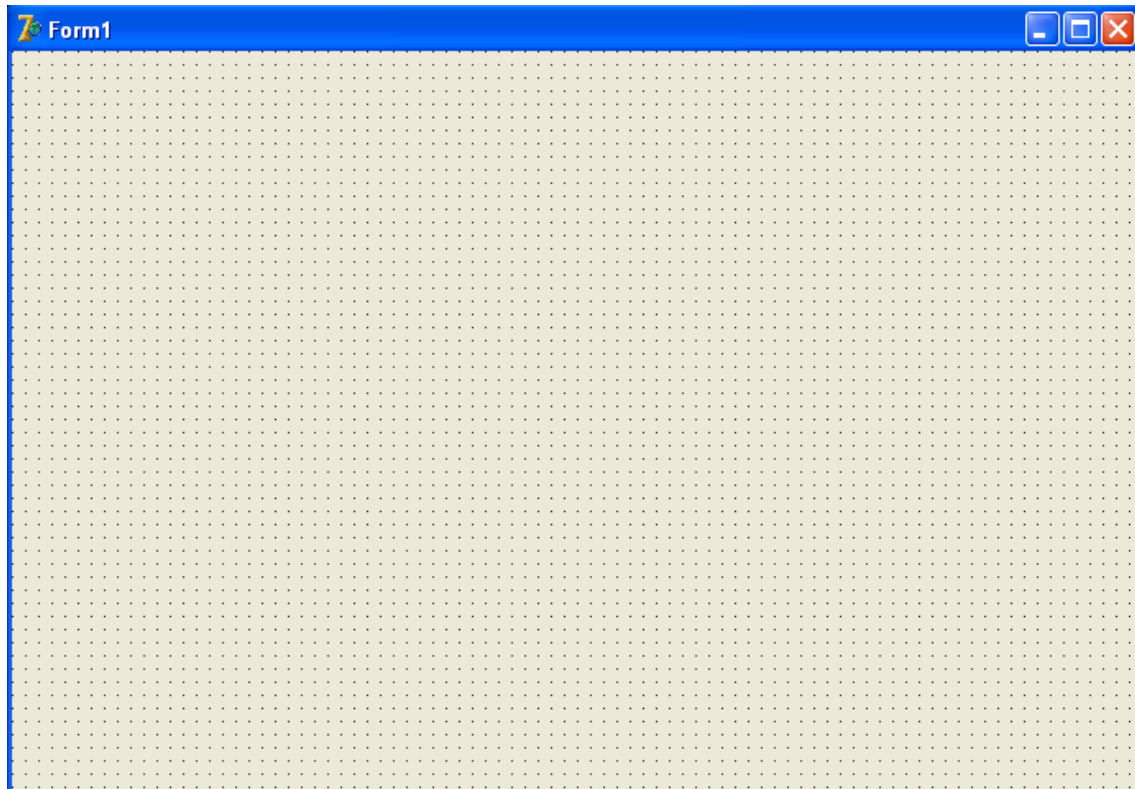
Bien, primeramente, en teoría lo que nuestro programa va a hacer es que al pulsar un **Botón** muestre un mensaje que diga **Hola Mundo**, para eso haremos que lo muestre mediante la función **ShowMessage**, que trae Delphi incorporada. Esta función se puede realizar mediante la **Unit Dialogs** que se declara en la Cláusula **Uses**. Pero para nosotros no es necesario, Delphi lo declara automáticamente, en resumen, Delphi hace el trabajo sucio. Bien, primero que todos aprenderemos a abrir el Delphi. Si ya lo tenemos instalado, vamos a **Inicio**, hacemos click en **Programas** o **Todos los Programas**, luego vamos al submenú **Borland**, y hacemos click en **Delphi 7**.

Luego que cargue la aplicación veremos una pantalla así:



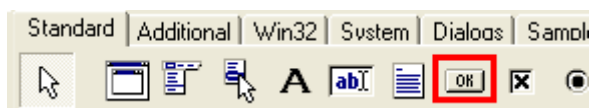
Lo que vemos a la izq. y arriba ya lo explicamos en capítulos anteriores, ahora les explicare lo que vemos en el medio.

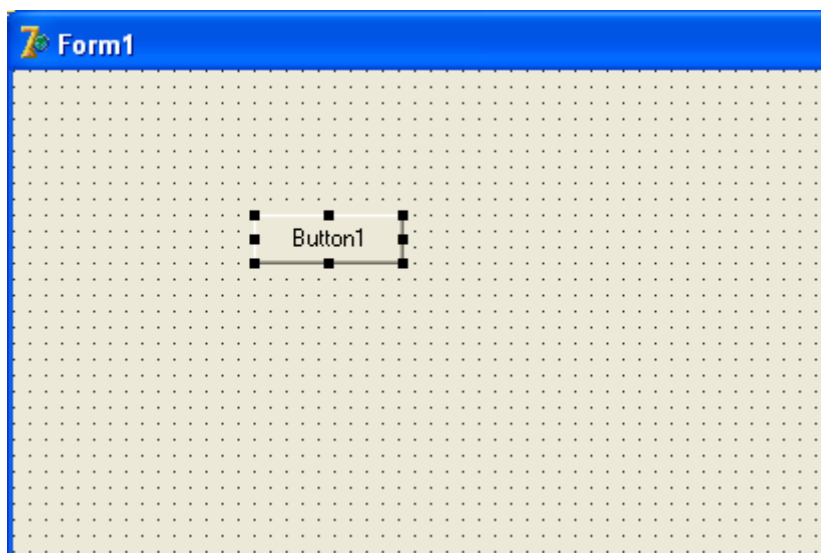
5.1.2 Formulario



Quizás para algunos, esto parece una ventana común y corriente con puntitos en el medio, pues están en lo cierto, pero se debe decir que es un ventana en fase de diseño, los puntitos forman una rejilla (grid). Esta ventana tiene un botón de Cerrar, Maximizar, Minimizar, un Icono (Icon), y un Nombre (Caption).

Para comenzar, añadiremos un Botón, cuyo componente es **Button**, buscamos el siguiente icono en la paleta **Estándar** del VCL, y hacemos click en el formulario.





Luego de hacer click en el Formulario, veremos que se crea una especie de Botón, como los de Windows 98, con el Nombre **Button1**. Generalmente cuando creamos un Componente se crea con el nombre del componente y el numero de componente de este tipo que el representa, por ejemplo, si añadimos otro Botón veremos que su nombre dice **Button2**.

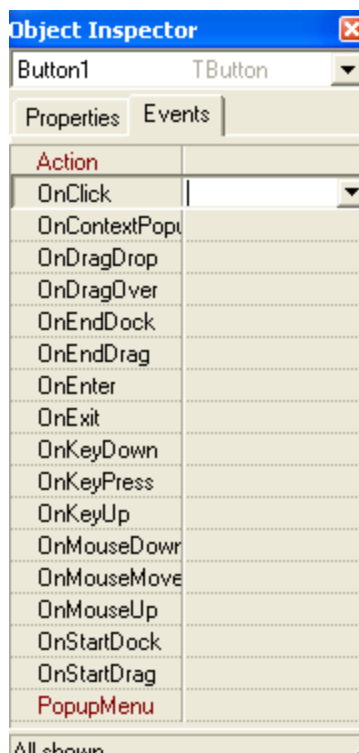
Ahora Voy a explicarles algo que muchos principiantes se preguntan,

¿Caption = Name?

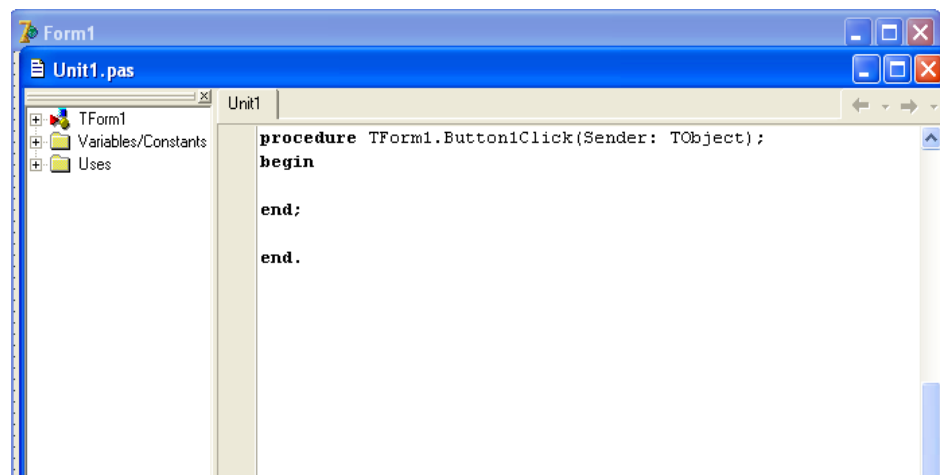
Todos los componentes tienen una propiedad llamada **Name**, que es la propiedad que les permite identificarse de los demás componentes, esta propiedad se puede cambiar para facilitarnos la etapa de programación del código, por ejemplo: tenemos una aplicación con muchos botones, entonces nos damos cuenta que seria muy engorroso trabajar con 30 botones identificándolos como Button1, Button2, Button3, etc., entonces podemos cambiarle el nombre a estos botones ha algo mas lógico y fácil de recordar, si tenemos un botón que lo que haga es cerrar el programa, le podemos cambiar la propiedad Name a **BotonCerrar** .

Ahora, la propiedad **Caption** se refiere al texto que va mostrar un componente visual que muestre un texto al usuario, por ejemplo el botón que insertamos dice Button1, pero si en la propiedad Caption ponemos Hola Mundo, veremos que dice Hola Mundo en el formulario, pero sin embargo no cambio la propiedad Name no cambio, el Boton Hola Mundo, sigue llamándose Button1, y la hora de programar código, nos debemos referir a el como Button1.

Bien, continuemos con el programa Hola Mundo, ya pusimos el botón, pero ahora donde ponemos el código que nos permitirá mostrar el mensaje, bien ahora seleccionamos el Button1, y en la ventana Object Inspector hacemos click en Events, luego veremos algo así:



Ahora, vemos que hay mas eventos que los que explicamos en el apartado eventos, pero esos los explicaremos mas tarde, ahora nos interesa el evento **OnClick**, porque la aplicación nos va a mostrar el mensaje cuando hagamos click sobre el Botón, entonces hacemos doble click en el campo de texto al lado del evento on click, y nos aparecerá el Code Explorer, algo que nunca esta cerrado, solo estaba detrás del **Form1**



Ven que dentro del mismo se crea un Procedimiento que hace referencia al evento onClick del Button1, que esta en el Form1.

Procedure TForm1.Button1Click(Sender: TObject);
Begin

End;

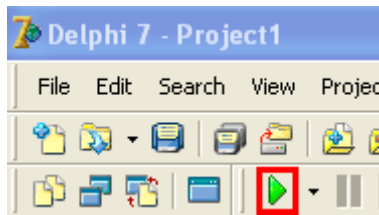
Bien aquí dentro de ese evento le diremos a la aplicación que Muestre el Mensaje "Hola Mundo":

Lo haremos de la siguiente manera, dentro del Begin y el End, escribiremos lo siguiente

ShowMessage('Hola Mundo');

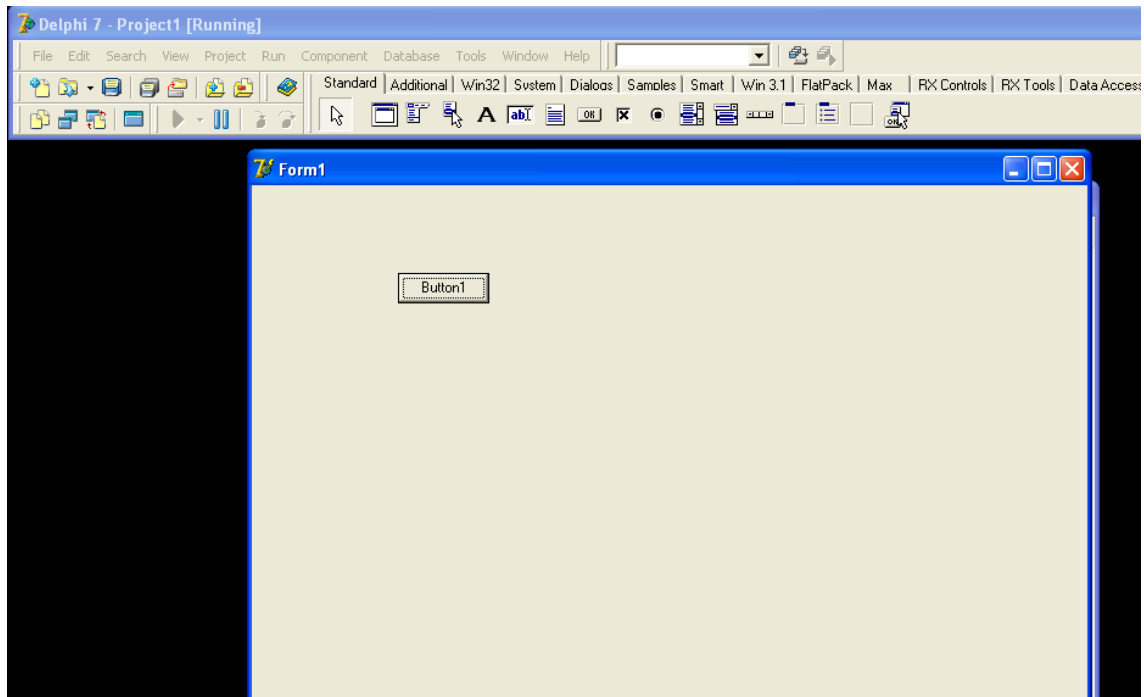
Bien, explicare detalladamente la función ShowMessage, esta función lo que hace muestra un dialogo igual a los de Windows, con el botón Aceptar, y un botón de cerrar, el nombre de este dialogo será el nombre de la aplicación, en este caso **Project1**, porque aun no lo hemos declarado.

Para compilar nuestro programa hay diferentes maneras, una, haciendo click en el botón **Run** en la barra de herramientas

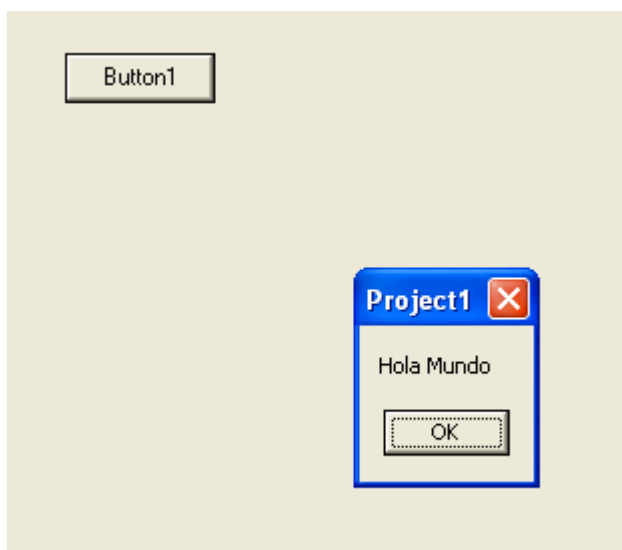


O Bien presionando la tecla **F9**.

Luego de esto veremos lo siguiente:



Veamos que los puntitos desaparecieron, bien, les habia dicho que anteriormente el formulario estaba en forma de diseño, ahora esta corriendo como una aplicación, o lo que se dice en Ingles **RunTime**. Ahora si hacemos click en el Botón, veremos un mensaje diciendo Hola Mundo



☺ Muy bonito, aunque no debemos presumir si logramos hacer esto, técnicamente, solo les he enseñado las bases de la programación en Delphi.

5.1.2.1 Algunas Propiedades de los Formularios

Antes de seguir con este curso debería explicarles algunas propiedades de los formularios que pienso yo le harán el trabajo mas fácil.

Align: Propiedad de Alineamiento, lo que significa que su valor le dirá al Delphi, en que posición justificada de la pantalla saldrá este formulario. Este puede tomar los siguientes Valores:

alBottom = Parte Baja de la Pantalla
alCustom = Parte en que estaba en la fase de diseño
alClient = Toda la Pantalla
alNone = Parte en que estaba en la fase de diseño
alTop = Parte alta de la Pantalla
alRigth = Parte derecha de la Pantalla
alLeft = Parte izquierda de la Pantalla

AlphaBlend: Esta propiedad puede ser True o False, lo que significa que si esta en falso no se tomará en cuenta, pero de lo contrario esto significaría que el Formulario y todo en su interior será medio transparente de acuerdo al valor de la propiedad que le Sigue **AlphaBlendValue**, esta trae un valor numérico, por defecto trae el 255, que es el máximo que admite, es decir, si esta en 255, el formulario se verá normal, si es menor se ira haciendo transparente la imagen.

BorderIcons: Es una propiedad que como su nombre lo dice son los iconos del borde, es decir los iconos del área de cerrar, minimizar, maximizar un formulario. Este trae cuatro propiedades internas:

biMaximize: Botón de Maximizar
biMinimize: Botón de Minimizar
biSystemMenu: Indica si mostrar el submenú(se explica mas adelante) al hacer click derecho en el icono de la aplicación, o en la barra de titulo. Además elimina los botones de Maximizar, Cerrar, y Minimzar
biHelp: Botón de Ayuda que traen algunas ventanas de Windows, cuando lo pulsas el cursor te cambia a una flechita con un signo de ayuda.

5.1.3 Mensajes

Bien, ahora quisiera explicarles algo mas sobre los Mensajes. Para el ejemplo Hola Mundo utilizamos la función:

```
ShowMessage('String');
```

Te habrás fijado que no te deja otra opción que contestar, porque sino este hará caso omiso. Esa característica de los cuadros de mensajes se llama **Modal**, y en general es aplicable a todas las ventanas. Cuando se activa una ventana modal el programa que invocó a esa ventana se detiene en espera de que se cierre esta, y si es el caso devuelva un resultado, que normalmente es la acción del usuario.

Pero sin embargo podíamos haber utilizados otras funciones mas complejas, pero a su vez mas serias..

Los tipos de cuadros de mensajes que existen son:

- 1. ShowMessage
- 2. ShowMessagePos
- 3. MessageDlg
- 4. MessageDlgPos
- 5. MessageBox

1. ShowMessage

ShowMessage es el más sencillo de todos ellos. El cuadro aparece por defecto en el centro, el título del mismo es el nombre de la aplicación, el texto lo indicamos nosotros, y tiene solo un botón.

2. ShowMessagePos

Esta tipo es igual al anterior pero tiene dos detalles más, que son dos valores que indican las coordenadas donde se mostrará el cuadro. Para indicar unas coordenadas dentro de la pantalla tienes que conocer el tamaño de la misma. Estas coordenadas son Alto y Ancho. (Height y Width). Respectivamente.

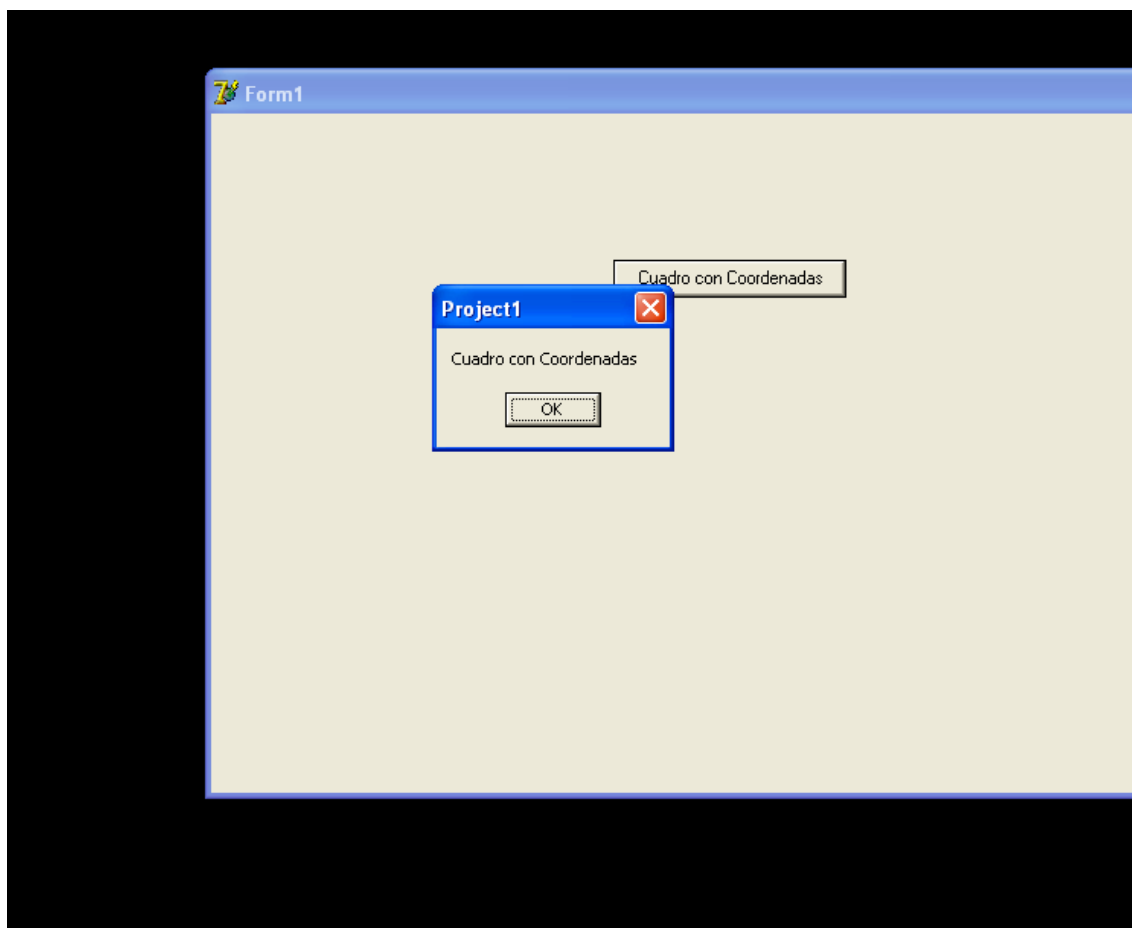
Pues a lo que iba, si pones Screen.Height, obtienes el valor del alto de la pantalla, y lo mismo para Screen.Width pero para el ancho. Así que el ShowMessagePos necesita que le indiques el texto, y además las coordenadas, primero el ancho y luego el largo. Añade un nuevo botón al proyecto y en su evento doble clic pon esta línea:

begin

```
ShowMessagePos('Cuadro con Coordenadas', Screen.Width div  
3,Screen.Height div 3);
```

End;

Veremos un resultado como este:



Sin embargo, deben haber notado algo que no ha sido explicado en ese código, si de hecho hay algo que no explique, el uso del comando **DIV**, que significa **Dividir**, en resumen lo que hacemos con `Screen.Width DIV 3`, es Dividir por 3 el ancho de la pantalla por 3.

3. MessageDlg

A continuación el MessageDlg, complicadito pero muy eficaz.

El primer dato que recibe la función es el texto que deseamos mostrar, luego el tipo de cuadro que mostraremos según una lista predefinida por Windows, el tercero es son los botones que se van a mostrar, y para terminar el cuarto es índice que esta relacionado con la ayuda de la aplicación que creamos si colocamos un botón de ayuda en el cuadro.

Los tipos de cuadro que exciten están indicados en la lista que a continuación muestro, ten en cuenta que según el tipo de cuadro Windows pinta un icono en tu cuadro.

- `mtInformation`
- `mtWarning`
- `mtError`
- `mtConfirmation`
- `mtCustom`

Los tipos de botones que podemos poner son los siguientes:

- mbYes
- mbNo
- mbOk
- mbCancel
- mbAbort
- mbRetry
- mbIgnore
- mbAll
- mbHelp

Cuando pones los tipos de botones que deseas debes ponerlos entre corchetes (**[]**), y seguidos de una coma. Otra opción es poner un serie de botones predefinidos, pero si lo haces **no** pongas los corchetes. Los tipos de botones predefinidos son:

- mbYesNoCancel
- mbAbortRetryIgnore
- mbOkCancel

Para probar esto pon el siguiente código en el evento onClick de un Boton:

```
If MessageDlg ('¿MessageDlg?', mtwarning,[mbyes,mbno],0) = mrYes then  
ShowMessage('Si')  
else  
ShowMessage('No')  
end;
```

Observarás el sgte. Resultado



Veras que si pulsas Yes, usando la opción ShowMessage mostramos un mensaje que diga Si, o de lo contrario No. Esto significa que si pulsamos si podemos decirle que ejecute cualquier código.

Observa que aquí hay más novedades, y es que MessageDlg devuelve el valor del botón pulsado por el usuario. El valor devuelto es una constante que puede tomar los siguientes valores:

- mrNone

- mrAbort
- mrYes
- mrOk
- mrRetry
- mrNo
- mrCancel
- mrIgnore
- mrAll

Observa que el valor de la constante tiene el nombre del botón pulsado, así en el ejemplo cuando es pulsado el botón mbYes, el valor devuelto es mrYes. Ten cuidado no vayas a esperar la pulsación de un botón que nos has puesto en el cuadro, porque nunca te devolverá ese valor la función.

4. MessageDlgPos

La variante de MessageDlg es MessageDlgPos, que es igual pero tiene un par de datos más, que es, al igual que ShowMessagePos, las coordenadas donde se situará el cuadro cuando se muestre. Su formato es:

begin

MessageDlgPos ('texto' , tipo de cuadro, botones, índice de ayuda, coordenada X, coordenada Y);

End;

Te habrás dado cuenta que todos estos cuadros están en inglés, pero tenemos la opción de ponerlos en castellano. Para eso utiliza la opción sgte.:

5. MessageBox

Application.MessageBox('Mensaje','Titulo',mb_okCancel+mb_IconExclamation);

El código de arriba muestra un resultado como este:



Fíjate , primero el mensaje a mostrar, luego el titulo, después los botones que queremos poner, y el icono también a nuestra voluntad.

Ahora, se preguntaran, porque este cuadro muestra el mensaje de los botones en español (aceptar y cancelar), y no en ingles (Yes y No) como los demás; pues muy fácil, el Cuadro de Mensajes que se llama usando el código `Application.MessageBox` hace una llamada al API de Windows, el cual le dice que Windows esta en Español, y le da el texto predeterminado para los botones

Ahora, para comprobar la respuesta de un botón, utilizaremos los sgtes. Códigos.

- `IdAbort`
- `IdCancel`
- `IdIgnore`
- `IdNo`
- `IdOk`
- `IdRetry`
- `IdYes`

Los botones que puedes utilizar son solo las combinaciones de botones predefinidas que listé más arriba. Y los tipos de iconos son:

- `Mb_IconAsterisk`
- `Mb_IconError`
- `Mb_IconExclamation`
- `Mb_iconHand`
- `Mb_IconInformation`
- `Mb_IconQuestion`
- `Mb_IconStop`

- Mb_IconWarning

Ejemplo de Comprobación:

Case...Of

Utilizando el método **Case Of** que significa, **en caso que esto haga esto**. Para especificar luego del **Of**, que hacer cuando se pulse cierto botón.

begin

case

Application.MessageBox('Mensaje','Titulo',mb_okCancel+mb_IconExclamation) **of**

ID_OK: ShowMessage('Aceptar');

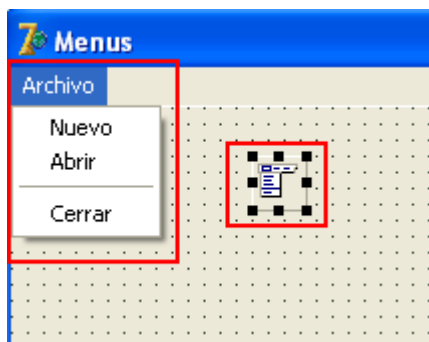
ID_CANCEL: ShowMessage('Cancelar')

end;

end;

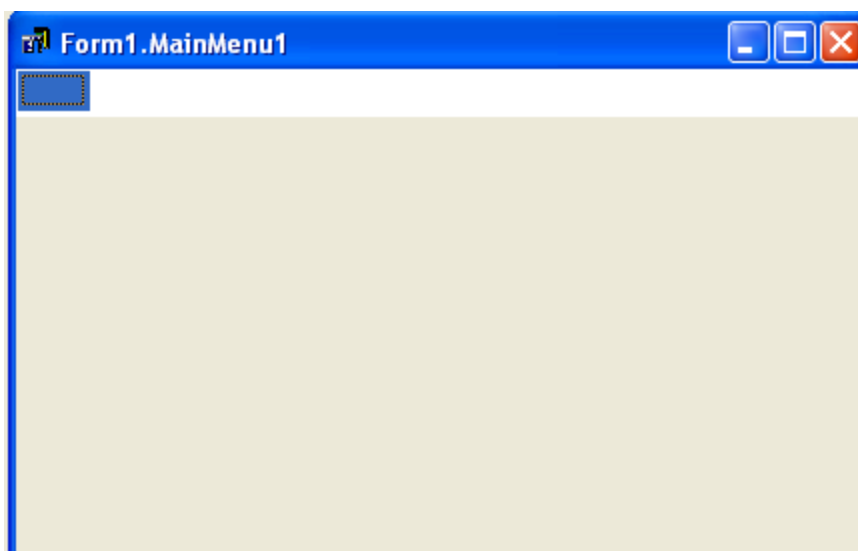
6.0 Conociendo Componentes

6.1 Los Menús (MainMenu)



Bien, así es como se ve un menú en la mayoría de las aplicaciones informáticas hoy en día, estos menús que tanto nos ayudan en la mayoría de los programas, contienen gran parte de los comandos para el uso y personalización del programa.

Delphi pone a nuestra disposición una herramienta para facilitarnos la tarea, además de tratar cada menú o submenú como un si fuese un componente más. Para empezar a trabajar con los menús, lo primero es colocar un componente TMainMenu, en el formulario. Así que desde la página Standard de Delphi, pincha el segundo componente de izq. a der.; luego hacemos click en el lugar del formulario donde queremos agregarlo, se creará un componente llamado **MainMenu1**, cuyo icono en el Form será el que está circulado en rojo en la imagen de arriba. Si hacemos doble click sobre él veremos algo parecido a esto.



Bueno, esta es la fase de diseño del Menú. Delphi nos facilita esta tarea, porque sería muy difícil ir declarando por código cada uno de los ítems del Menú.

Bien, vamos a crear un menú simple, bueno al hacer doble click en el icono del Menú nos encontramos con la imagen de arriba, y se encuentra seleccionado un rectángulo, el mismo es el primer ítem del Menú, es decir, es un componente que tiene propiedades, por lo que en el Object Inspector buscaremos su propiedad Caption, y pondremos el nombre que quieras, en este caso por **&Archivo** bien, te preguntarás porque **&...**, este símbolo, Delphi lo reconoce como la rayita que subraya a una letra en los menús, para poder navegar por los mismos usando la tecla ALT. Al escribir **&Archivo** y pulsar ENTER, se crea automáticamente un ítem debajo de él que ya teníamos, en el cual vamos, con el mismo procedimiento a escribir **&Salir**, para poder salir del programa mediante ese Menú. Bien, si hacemos doble click en el Menú Salir, (Salir1) nos saldrá el Code Explorer, para poner un código refiriéndose al evento OnClick, en este caso usaremos una función para cerrar el programa.

Para esto existen varias formas. Si el formulario a cerrar, es el formulario principal de la aplicación, basta con decir al evento

Begin

Form1.Close; // Form1, puede variar de acuerdo al nombre del Formulario

End;

En caso de que no sea el formulario principal Podemos decir:

Begin

Application.Terminate; // Le dice a la Aplicación que termine

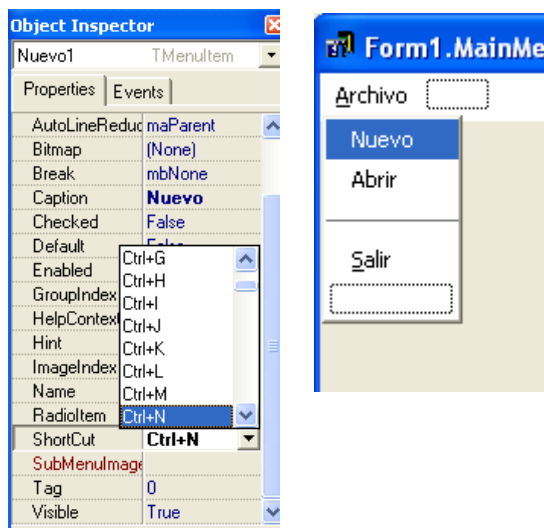
End;

Ahora ejecuta tu aplicación y verás que al pulsar sobre el menú Archivo, se te desplegara un menú, que dirá Salir. Si pinchamos aquí, se cerrará la aplicación y volverás al ambiente de diseño de la aplicación

ShortCuts

Supongo que sabes que es un ShortCut, es un atajo por el teclado a una acción predefinida por el programador para una acción.

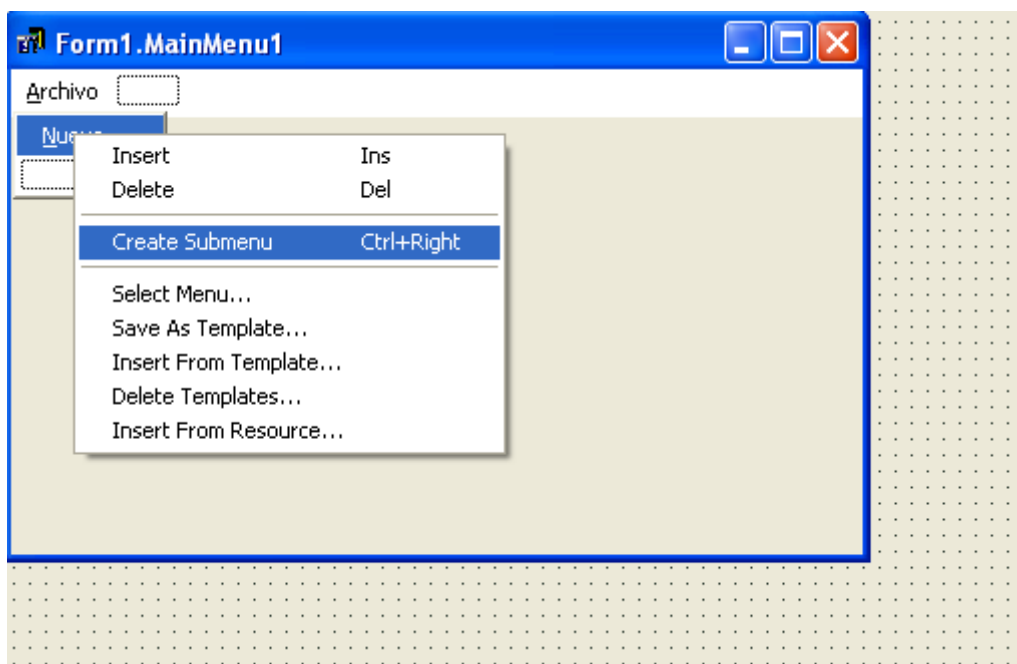
Bueno Delphi nos pone a nuestra disposición una propiedad en los menús para eso. Seleccionamos el Ítem del Menu para hacerle el ShortCut, y luego en el Object Inspector buscamos la propiedad ShortCut, y le ponemos el valor que queremos.



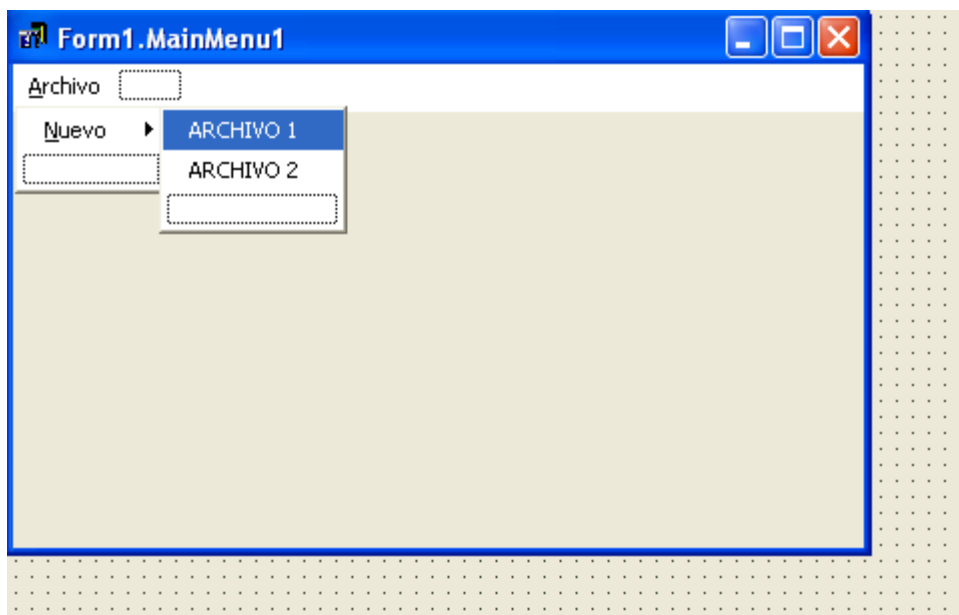
Hint: Si quieres crear en un menú un separador (una rayita que separe un ítem de otro), solo basta crear un ítem que cuyo Caption sea " - " un guión. Y verás que en la fase de diseño saldrá un rayita larga.

6.1.1 Los Submenús.

El trabajo con los submenús es igual de los menús, solo que para crearlos debes hacer click derecho y verás algo así:

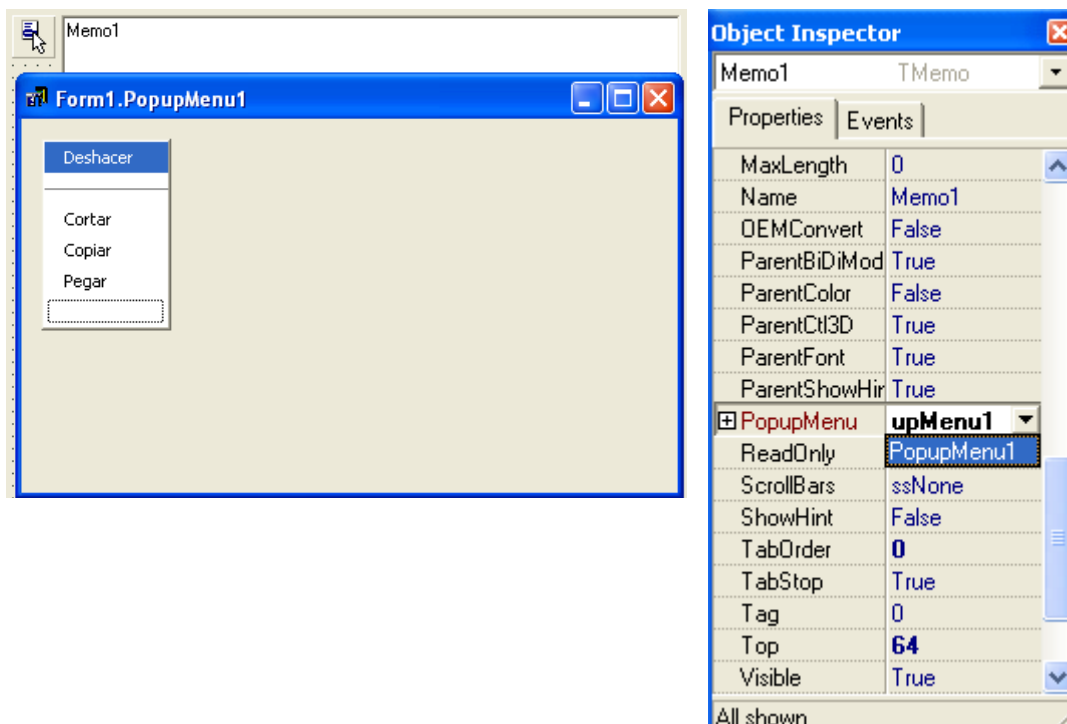


Entonces haces Click en el Menú **CREATE SUBMENU**, y automáticamente se crea un submenú, el cual tratas igual que un Menú.

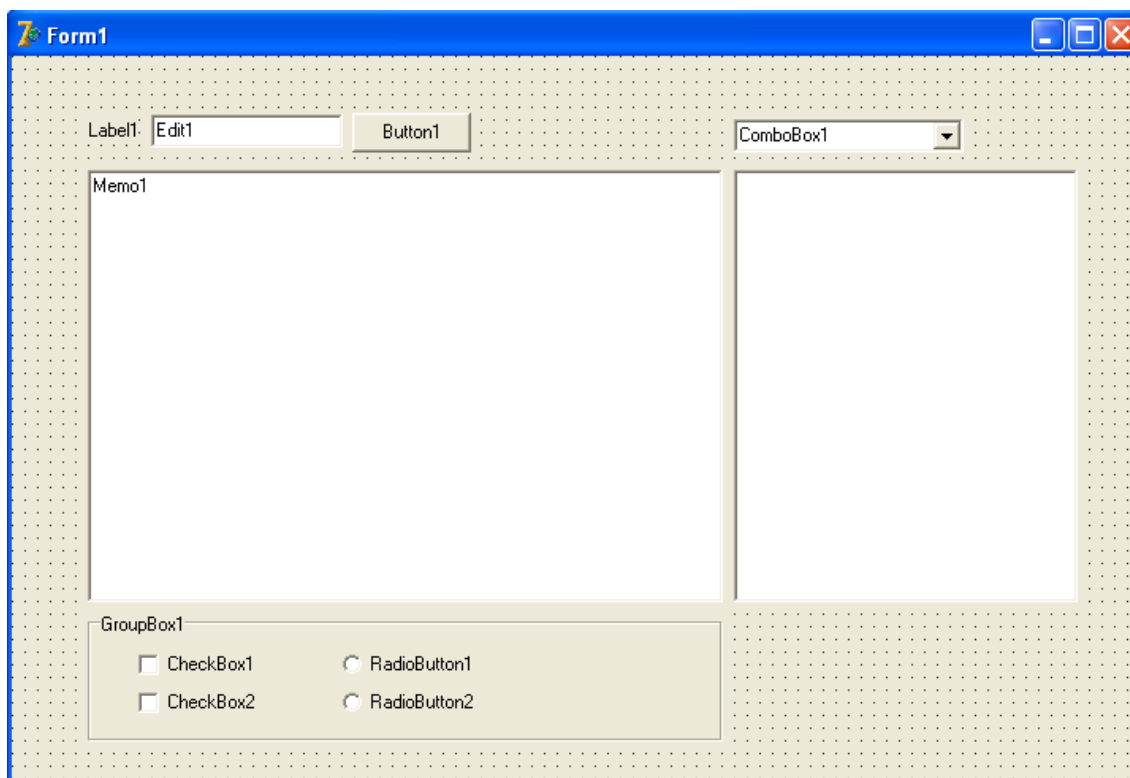


6.1.2 Menús Emergentes (PopUpMenu).

Todo lo que te he contado hasta ahora es aplicable a un menú tipo emergente (PopUp), son iguales, pero lo único que los diferencia es que los menús emergentes se despliegan al pulsar sobre un control con el botón derecho del ratón. Pues para usar un menú de este tipo lo único de que debes hacer es colocar un control de este tipo sobre el formulario que contiene el control que hará uso del menú emergente (puede ser casi cualquier control, o el mismo formulario), configúralo como si se tratase de un menú normal, y ahora en el control que deseas que tenga este menú emergente, selecciónalo en su propiedad PopUpMenu, y ya está. Prueba tu programa y pulsa sobre el componente o formulario con el botón derecho del ratón.



6.2 Otros Componentes



La imagen de Arriba muestra otros componentes muy útiles del Delphi, todos están en la paleta Estándar:

Label1: Este componente es una etiqueta, nos sirve para poner texto en nuestro formulario

Edit1: Este componente es un cuadro de texto, nos sirve para escribir texto en Runtime y luego poder utilizarlo para otros Fines.

Memo1: Es un cuadro de texto para editar ficheros .txt, o para mostrar un texto largo.

ComboBox1: Es como una lista de varios cadenas de textos que se almacenan en un menú desplegable.

ListBox1: Es el cuadro blanco abajo del ComboBox1. Este es una lista de cadenas de textos pero se muestran todas como si fuera una tabla de una columna (Se puede codificar para 2 o más columnas)

GroupBox1: Componente que sirve para poner dentro de el componentes que se identifiquen para una cosa

CheckBox1, CheckBox2 : Casillas de verificación que al presionarlas se marcan con una X. Esenciales para editar propiedades.

RadioButton1, RadioButton2: Botones de opción, cuando presiones uno se desmarca el otro siempre y cuando el otro este en el mismo GroupBox, o en un formulario. Si tienes dos GroupBox en un formulario y dos RadioButton en cada uno de ellos, la selección de un RadioButton de un GroupBox no interfiere en la de los RadioButton del otro GroupBox.

7.0 Nuestra primeras Aplicaciones.

7.1 Procesador de Textos

Bien, comenzaré a enseñarles como crear un procesador de textos simple, y luego otras cosas para que el mismo sea más complicado.

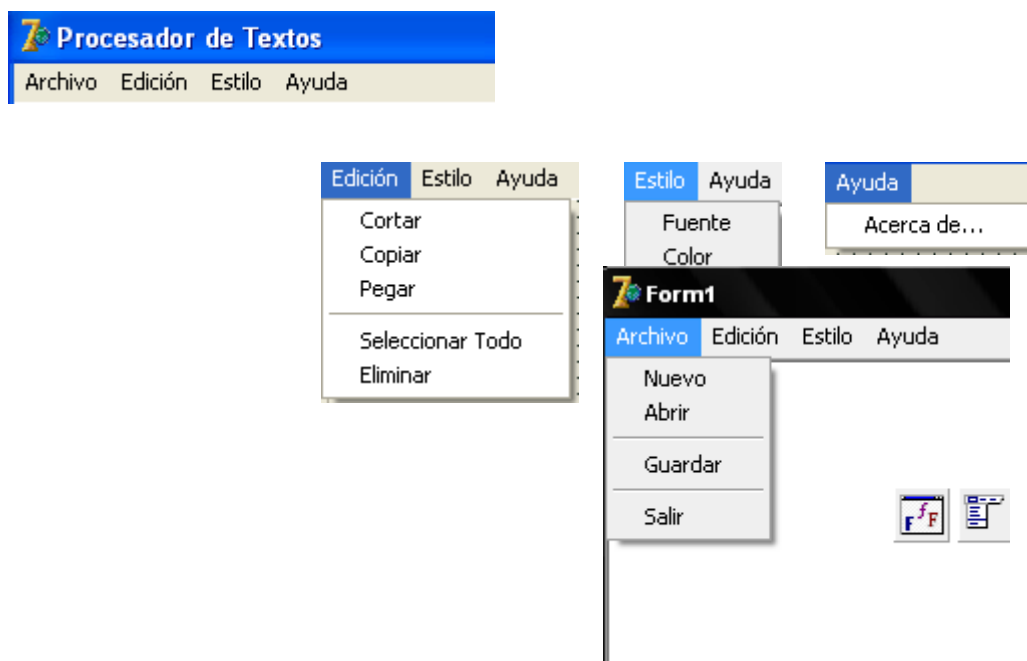
Un Procesador de textos, es una aplicación que permite al usuario crear y editar texto, guardarlo, abrirlo, etc. Nuestra aplicación constara de todas esas funciones, pero se las enseñare por partes.

Este procesador solo nos permitirá editar documentos con la extensión **txt**

Primero a la Propiedad **Caption** del formulario pónganle **Procesador de Textos**. Para hacer esto, seleccionen un espacio del formulario, y busquen en el *inspector de propiedades* la propiedad **Caption**, y escriban lo que les dije.

7.1.1 Apariencias y Menús

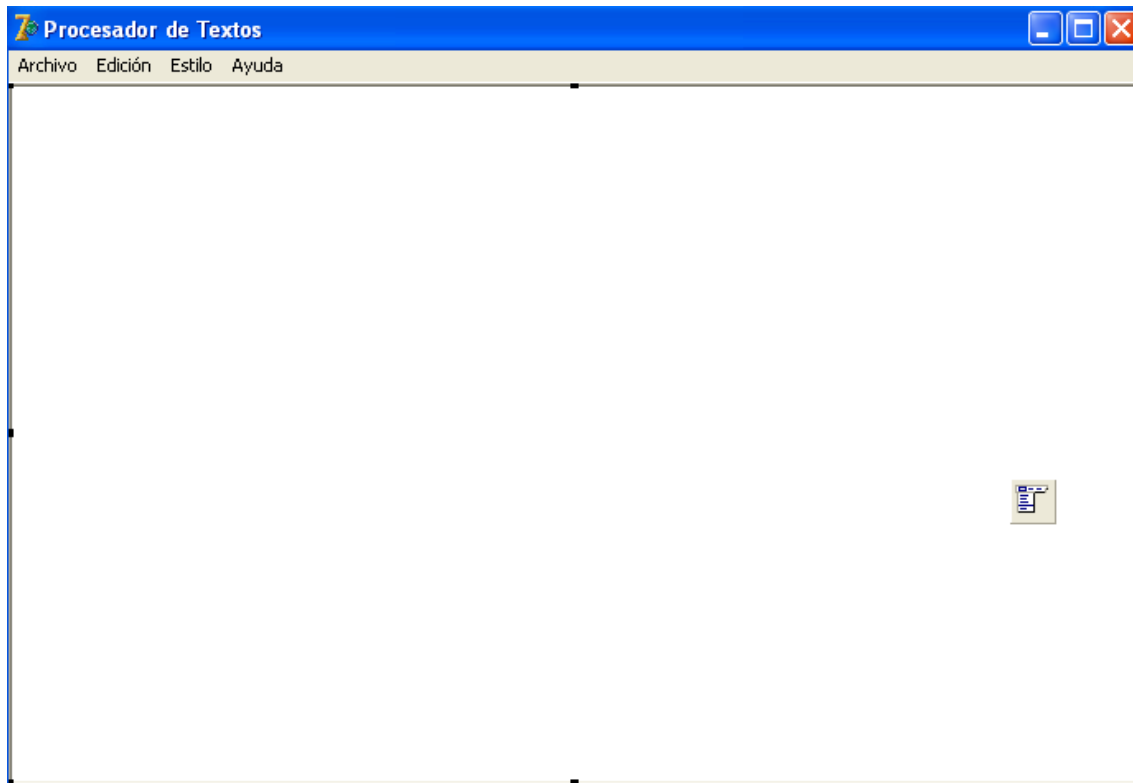
Primeramente insertaremos un Menú (MainMenu), esta en la paleta **Standar**, en su propiedad NAME, le pondremos **MenuPrincipal**. Crearemos un menú con la siguiente estructura.



Se darán cuenta de que hay cuatro Menús, y ninguno de sus Ítems tiene un ShortCut activado, lo he dejado así para que usted pueda ponérselos como usted quiera.


Bien ahora insertaremos un Componente Memo de la Paleta **Estándar**, que será donde se escribirá el texto. Ahora, en su propiedad **Align** pondremos **alClient**, para que cubra toda el formulario. Luego en la Propiedad Lines (del tipo StringList, que guardara una lista de string) la editamos haciendo clic en el boton (...) que aparece, y al salir el String List Editor le borramos lo que dice Memo1.

Luego de esto el formulario se vera así:



7.1.1.1 Archivo. Bien, ahora comenzare a explicarles por la parte más fácil.

Primero el Boton **Abrir** del Menú **Archivo**.

- Existen varias formas abrir un documento **txt**, en este caso utilizaremos un OpenFileDialog,  componente existente en la paleta **Dialogs**. Inserten uno en la aplicación.

Luego verán que tienen unas propiedades no muy comunes.

Properties	Events
Ctl3D	True
DefaultExt	
FileName	
Filter	
FilterIndex	1
HelpContext	0
InitialDir	
Name	OpenDialog1
Options	[ofHideReadOnly, of
OptionsEx	[]
Tag	0
Title	

Les explicaré las siguientes:

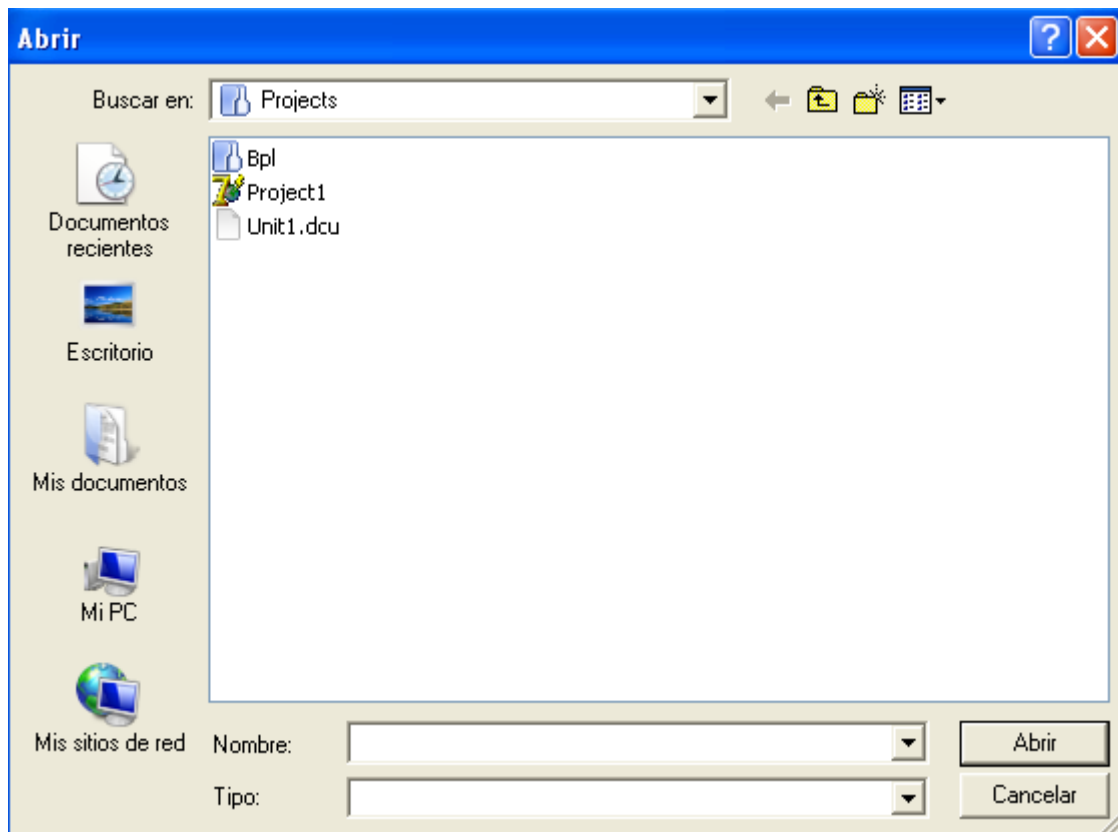
FileName: Le dice a la aplicación cual será el archivo al cual hará referencia este diálogo.

Filter: Edita las extensiones de archivo que el dialogo va a buscar en RunTime.

InitialDir: Especifica cual será el directorio que se abrirá cuando se cargue el OpenFileDialog

Title: Indica el titulo que tendrá el Dialogo, si lo dejas en blanco solo mostrara el texto **Abrir**.

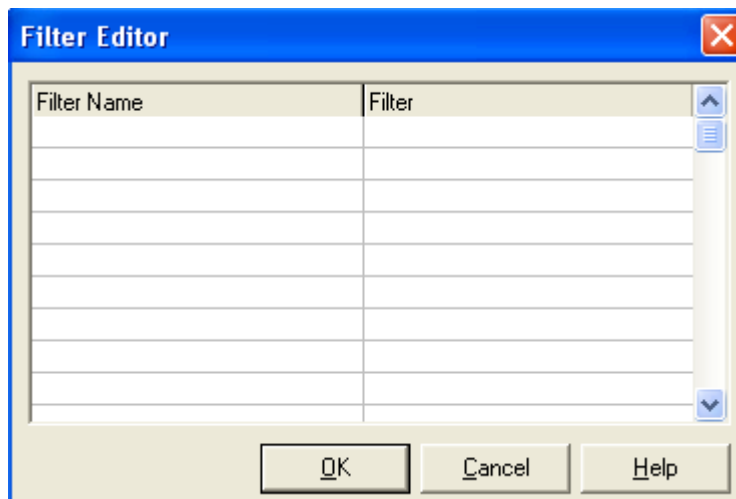
Un dialogo se ve así:



Bien, en nuestra aplicación utilizaremos las propiedades:

FileName, Filter y Title


Primeramente vamos a la propiedad Filter (...), y se nos abrirá una pantalla llamada Filter Editor (Editor de Filtros).



Donde dice Filter Name irá el texto que se va a mostrar en el ComboBox Tipo del Dialogo, y Filter, las extensiones que va a mostrar el.

Ahora pondremos lo siguiente:

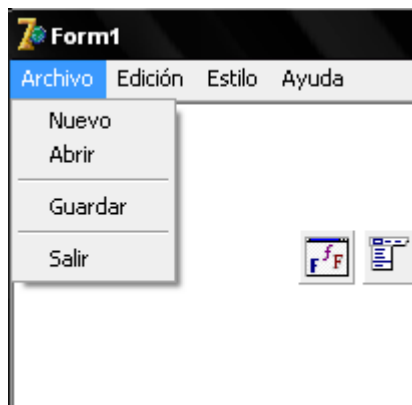
Filter Name	Filter
Documentos de Texto (*.txt)	*.txt
Todos los archivos (*.*)	*.*

Luego insertaremos el Componente **SaveDialog**, , también en la paleta **Dialogs**, que sus propiedades casi igual a las del **OpenDialog**, este componente nos servirá para guardar el documento, y en su propiedad Filtro pondremos la siguiente estructura

Filter Name	Filter
Documentos de Texto (*.txt)	*.txt
Todos los archivos (*.*)	*.*

Bien en el menú archivo existe un elemento llamado **Nuevo** el cual como todos saben en Procesador de Texto lo que hace es crear un nuevo documento, o sea borrar lo que tiene el **memo** escrito.

Entonces en el evento **OnClick** del elemento nuevo, o sea hacemos click en **Nuevo** en edición:



Entonces, esto te abrirá el Explorador de códigos con el cursor en la línea autocreada para insertar o escribir código para el evento **OnClick** del elemento nuevo.

Para hacer esto pondríamos lo siguiente:

Begin

Memo1.Clear; //Limpia el memo

End;

Para programar el elemento guardar, hacemos click en el elemento guardar para editar su evento **OnClick**. Lo que haremos será utilizar el Componente **SaveDialog1** que ya hemos insertado, y configurado para el procesador de texto.

El procedimiento se vería así

begin

If SaveDialog1.Execute **then**

Memo1.Lines.SaveToFile(SaveDialog1.FileName+'.txt');

end;

Con lo anterior lo que hacemos es Que si ejecutamos el SaveDialog1 (En este caso lo estamos ejecutando con la primera línea), vamos a hacer lo que le sigue, utilizando la condición **If Then**. Entonces, como funciona....Mediante el uso de la función SaveToFile vamos a guardar las Lineas del Memo con el nombre escrito en la propiedad FileName del SaveDialog agregándole la cadena de texto .txt al final. Espero que hayan entendido (esto ocurre en la 2da línea, si saben algo de ingles se darán cuenta).

Ahora, porque hay que agregarle +'.txt' al final. Pues para ser sincero, lo que hago es que el Nombre del fichero (filename) le agrego la extensión, porque no se porque el SaveDialog no me la añade automáticamente aun declarándola en el Filter. Si alguien sabe que me escriba y que me diga.0

Ahora el Elemento **Abrir** seria parecido, solo con algunos cambios. Primero el Código luego la explicación

Begin

if OpenFileDialog1.Execute **then**

Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

end;

Primero ejecutamos el OpenFileDialog1 y luego mediante la segunda línea, Cargamos desde un fichero, las líneas del memo, El nombre fichero sería el que esta escrito en la propiedad FileName del OpenFileDialog. Lo pueden ver en el Lines.LoadFromFile (RUTADELFIHERO), la ruta en este caso es devuelta por el OpenFileDialog.FileName.

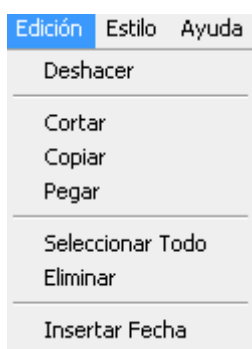
Ahora para salir del Programa utilizaremos la función **Terminate**. Que la usaríamos en el Elemento Salir del menú archivo. Esta función lo que haría sería Terminar la Aplicación

Begin

Application.Terminate;

End;

7.1.1.2 El menú **edición** será el siguiente. Con las Opciones



Estas opciones son de lo más fáciles así que no me voy a detener mucho en ellas. Espero que uds. Tengan conocimientos en Ingles, aunque sea el más mínimo. De todas formas, les aclaro algo:

Clipboard: Se refiere al portapapeles de Windows, si a ese espacio en memoria en donde se copia todo lo que le das Ctrl+C o Click Derecho Copiar. Eso es el Portapapeles (Clipboard)

Cut, Copy, Paste, SelectAll, Undo: Cortar, Copiar, Pegar, Seleccionar Todo, Deshacer respectivamente. Solo

Aclarando

Antes de poder hacer uso del portapapeles debemos decirle a la aplicación que se va a usar el Clipboard, para esto, añadimos **ClipBrd** a la clausula Uses de la unit, o sea...

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics,
Dialogs, StdCtrls, Menus, **ClipBrd;**

Luego haciendo click en los elementos del menú correspondientes agregamos los siguientes códigos:

Deshacer:

Begin

Memo1.Undo; //Deshacer cambios en el memo1

End;

Cortar:

Begin

Memo1.CutToClipboard; //Corta el texto seleccionado en el Memo al Clipboard

End;

Copiar:

Begin

Memo1.CopyToClipboard; //Copia el texto seleccionado en el Memo al Clipboard

End;

Pegar:

Begin

Memo1.PasteFromClipboard; //Pega el texto existente en el Clipboard

End;

Seleccionar Todo:

Begin

Memo1.SelectAll; //Selecciona todo el texto en el memo
end;

Eliminar:

Begin

Memo1.ClearSelection; //Borra la selección del memo

End;

Insertar Fecha:

Begin

Memo1.Lines.Add(DateTimeToStr(Now)); //Insertamos la fecha actual

End;

Ahora vamos explicar como funciona lo de insertar la fecha, aquí vemos dos cosas nuevas. Primero la Funcion **Add** del componente Memo (Aunque existe en muchos componentes también), que sirve para Agregar una línea al memo. Esta función se le puede especificar una línea, con contenido **String**; Lo segundo Nuevo, es la variable **Now**, que contiene con formato **DateTime** la Hora y Fecha actual del sistema. Entonces lo que vamos a hacer es insertar esa hora en el memo, o sea poner el contenido de la

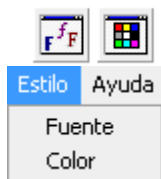
variable **Now** en el memo, pero que pasa, no podemos solamente poner `memo1.Lines.Add(Now);` ¿porque? Pues porque **Now** no esta en formato String, y nos dara error al compilar el programa. Entonces Hacemos uso de la función **DateToStr** (Date To String o Fecha to Cadena de texto), función que explicamos al principio de este curso, que sirve para convertir fecha a cadena de texto.

Ahora podemos compilar el programa y ver el resultado al pulsar en **Insertar Fecha:**



7.1.1.3 Estilo. Ahora explicaremos el menú estilo.

Este menú no es muy necesario, porque lo que va a hacer es cambiar el Color del Memo, y el tipo y tamaño de letra del documento, algo que no es muy necesario porque este no se guarda con el fichero .txt porque es texto sin formato. Esto les servirá para la hora de escribir que vean mejor el texto: Lo Primero que harán será insertar el Dialogo FontFialog y el ColorDialog (ambos en la paleta dialogs). El primero para la fuente, y el segundo para el color del memo.



Ya tenemos el menú creado, lo hicimos hace rato xDD. Bien solo nos queda programarlos, cosas que es muy fácil de hacer y de entender.

Primero el Elemento **Fuente:**

En el evento OnClick escribiremos lo siguiente

begin

If FontDialog1.Execute **then**

`Memo1.Font := FontDialog1.Font; //Igualando la fuente del Memo a la del FontDialog1`

End;

Primero ejecutamos el FontDialog1. Luego le decimos a la propiedad Font del Memo que será igual a la propiedad Font del FontDialog1. O sea que luego de seleccionar una fuente en el FontDialog1 se cambiara la fuente en el memo1.

Ahora el Elemento **Color:**

En el evento OnClick escribiremos lo siguiente

begin

If ColorDialog1.Execute **then**

`Memo1.Color := ColorDialog1.Color; //Igualando el color del Memo al del ColorDialog1`

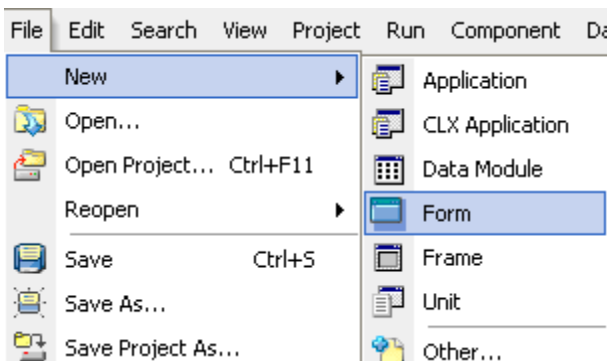
End;

Primero ejecutamos el ColorDialog1. Luego le decimos a la propiedad Colordel Memo que será igual a la propiedad Color del ColorDialog1. O sea que luego de seleccionar un color en el ColorDialog1 se cambiara el color en el memo1.

7.1.1.4 Ayuda. Ahora explicaremos el menú ayuda

Aquí en el Menú Ayuda solo crearemos un dialogo de ayuda Acerca de..., para que pongas los datos del Autor, versión e icono del programa. Estoy seguro que todos saben que es.

Primero crearemos un formulario ¿Cómo? Pues yo les explico, Vayan al Menú **File** del Delphi, submenú **New**, Form:



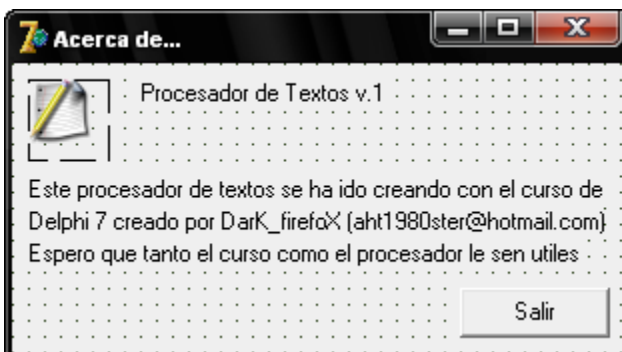
Bien, te saldrá un formulario vacío, en el cual vas a poner esa información. Utilizando los componentes Label e Image vamos a darle forma a este formulario. Primero en la Propiedad Caption del Formulario vamos a ponerle **Acerca de...** Luego Cambie el tamaño al formulario, pónganlo mas chiquito.

Bien... ahora vamos a insertar el texto que quieras con el componente label, que ya les explique al principio. Ahora, lo que es nuevo. El componente Image, que como su nombre lo indica Imagen, sirve para mostrar imágenes. Este componente lo podemos encontrar en la paleta Additional.



Ese que tiene el icono de un paisaje. Lo insertamos. En el formulario Acerca De...(form2)

Entonces en la propiedad Picture: Picture [None] ..., tocamos los 3 punticos, y buscamos la imagen que queremos mostrar (Tipo: All (*.jpg;*.jpeg;*.bmp;*.ico;*.emf;*.wmf)). Algo que sea relacionado con el Procesador De textos. Luego de usar esto, reajustamos el tamaño de el componente en el formulario, para ajustarlo al tamaño de la imagen. Luego de poner lo labels, el Procesador se veria algo, así aunque con tus datos:



Ese boton de salir, será para cerrar ese formulario, y volver al editor. En el evento **OnClick**, le pones lo siguiente:

Begin

Form2.Close; //Form2 es el nombre del formulario, y close para cerrarlo

End;

Ahora nos falta hacer para que al pulsar **Acerca de...** en el menú ayuda se muestre este formulario nuevo. Primero en la Clausula uses del formulario 1

agrega: **Form2**, para que este sepa que va tener código referente a este formulario.

Entonces en el Elemento **Acerca De...** pondríamos lo siguiente:

Begin

Form2.ShowModal; *//Mostrar el form2 como una ventana Modal*

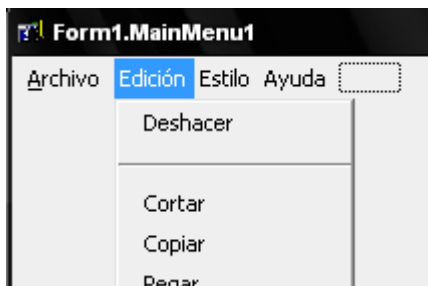
End;

ShowModal, no lo habíamos, visto, sirve Para mostrar el formulario 2 sin la posibilidad de activar la ventana del editor, es decir, el formulario 2 se lleva el foco total de la aplicación, o sea es el único permitido a estar activo.

7.2 Procesador de Textos

El procesador de Textos que hicimos era una introducción para que supiera como trabaja, ahora le vamos a hacer al mismo algunos arreglos.

7.2.1 Lo primero que haremos será, realizar una condición para el menú Edición, si son atentos se darán cuenta, que los elementos Cortar, Copiar, Pegar, Eliminar, están habilitada aun si no hay nada seleccionada en el Memo o en el caso de pegar que la Clipboard no tenga formato Texto. O sea vamos a hacer para que esto no suceda.



Primero, abriremos el editor del menú, ahora haremos Doble Click en el Menu Edición, exactamente donde dice Edición, para editar el evento OnClick del Elemento Edición

El código seria este:

var

seleccion : **Boolean**; *//Variable de tipo boolean(true o false)*

begin

if Memo1.Sellength <> 0 **then** *//Comprobamos el tamaño de la selección, si es mayor o menor que 0 entonces*

begin

Seleccion := true; *//Si es mayor que 0 entonces Selección=true*

end

else *//Sino*

Seleccion := false; *//La selección=false*

If Clipboard.HasFormat(CF_TEXT) **then** *//Comprueba si la Clipboard tiene format tipo texto*

Pegar1.Enabled:=True *//Le dice que se habilite*

else

Pegar1.Enabled:=False; *//Le dice que se deshabilite*

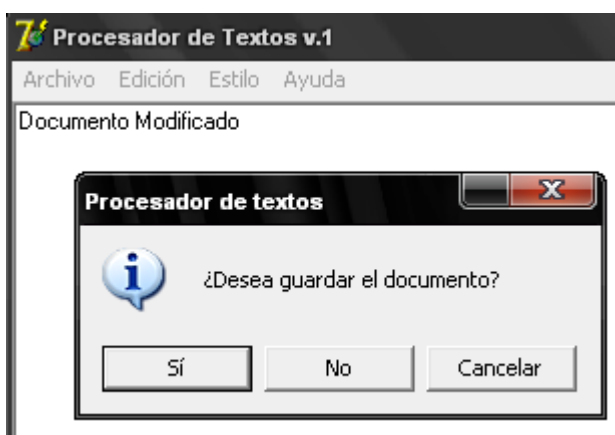
Cortar1.Enabled := Seleccion; *//Que va a estar habilitado de acuerdo al valo de Seleccion*

```
Copiar1.Enabled := Seleccion; //Que va a estar habilitado de acuerdo al valo  
de Seleccion  
Eliminar1.Enabled := Seleccion; //Que va a estar habilitado de acuerdo al  
valo de Seleccion  
end;
```

Aquí vemos varias cosas nuevas, primero **SellLength**, se refiere al tamaño de la selección, es decir, a la selección, y comprueba en este ejemplo si la selección es **Mayor > o Menor < que 0**. Creamos una variable de tipo Boolean, que ya se explico en que consiste. Entonces si el largo de la selección (**SellLength**) es mayor o menor que 0 (o sea si hay selección), vamos a decirle a la variable que tiene valor True, sino, le decimos que tiene valor False. Luego que se ejecute la condición le diremos a la propiedad **Enabled** de los elementos del menú (que es la que define si esta o no esta habilitado) que va a ser igual al valor de la variable de tipo boolean **Seleccion**. Ahora, tenemos también la función **HasFormat**, para comprobar si el ClipBoard tiene formato y hay que decirle entre paréntesis que tipo de formato va a buscar en el Clipboard, en este caso **CF_TEXT**, para buscar texto.

*Nota: Si quisiéramos buscar una imagen pudiéramos usar **CF_PICTURE** o **CF_BITMAP***

7.2.2 Lo segundo a añadirle a nuestro procesador de textos, es la posibilidad de prevenir la perdida de datos mediante un mensaje al usuario en caso de que se halla modificado el documento, diciéndole que si desea guardarlo antes de Crear un nuevo documento, cerrar la aplicación o abrir un documento nuevo. Seria algo asi:



En este caso, he modificado el documento, y le di a Nuevo en el menú archivo, y me mostro el mensaje de la imagen. Si le doy que Si, me sale el cuadro de guardar y luego me crea un nuevo documento en el memo (o sea le borra el contenido)

Entonces debemos cambiar el código del elemento **Nuevo** a:

```
begin
```

```
If Memo1.Modified then
```

```
Case Application.MessageBox('¿Desea guardar el documento?','Procesador  
de textos',Mb_YesNoCancel+Mb_IconInformation) of
```

```
ID_YES:
```

```
begin
```

```

guardar1.Click;
Memo1.Clear;
end;
ID_NO:Memo1.Clear;
ID_Cancel:
end;
end;

```

Explicare por partes:

If Memo1.Modified then

Esto revisa si el Memo ha sido modificado entonces ejecuta el código que le sigue

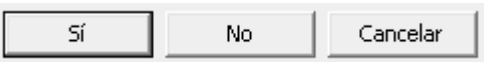

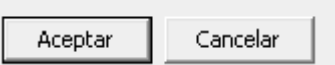


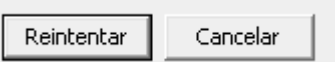
Case Application.MessageBox('¿Desea guardar el documento?', 'Procesador de textos', Mb_YesNoCancel+Mb_IconInformation) of

El **Case Of** es un tipo de condición. Que en este caso la utilizamos para el Cuadro de mensaje del Tipo **MessageBox** a nivel de aplicación. Este tipo de Mensaje permite mostrar una ventana con una serie de botones y devuelve un valor indicando el boton que pulso el usuario.








El primer parámetro: *¿Desea guardar el documento?* Se refiere al texto que mostrara el Mensaje. Es de tipo STRING

El segundo Parámetro: *Procesador de textos* Se refiere al nombre de la ventana del mensaje. ES de tipo STRING

El tercero: *Mb_YesNoCancel+Mb_IconInformation* Se refiere a la cantidad de botones y el icono que mostrara el mensaje respectivamente. Este es la mas amplia. Este posee varias combinaciones, si no queremos iconos no ponemos *+Mb_IconInformation*.

Botones	
MB_YesNoCancel	
MB_Ok	
MB_OkCancel	
MB_AbortRetryIgnore	
MB_YesNo	
MB_RetryCancel	

Iconos

Mb_IconInformation	
Mb_IconQuestion	
MB_IconHand (con sonido)	
MB_IconAsterisk (con sonido)	
Mb_iconWarning (con sonido)	
MB_IconError (con sonido)	
MB_IconStop (con sonido)	

Ahora, cuando nos referimos a:

ID_YES:

begin

guardar1.Click;

Memo1.Clear;

end;

ID_NO:*Memo1.Clear;*

ID_Cancel:

Esto nos servirá para decirle a la aplicación que hacer de acuerdo al boton que ha pulsado el usuario. Nos referimos a cada boton por su nombre antecedido por el prefijo **ID_**, o sea si queremos hacer algo al pulsar el boton YES (SI), seria **ID_YES: lo que queremos hacer;** y así sirve para cualquier boton.

Siendo esta la Opcion de guardar en caso de que se diga que si se quiere guardar, Primero hacemos el procedimiento Guardar, contenido en el evento onClick del elemento guardar (guardar1) del menú archivo, y luego borramos el contenido del memo. Si es que no se quiere guardar, solamente borramos el contenido del memo, y si el usuario le da cancelar no hacemos nada.

Ahora para la opción de **Abrir** y **Salir** nos sirve casi el mismo código, aunque con algún cambio en lo que tiene que hacer.

En el elemento **Abrir** pondríamos lo siguiente:

begin

If Memo1.Modified **then**

Case Application.MessageBox('¿Desea guardar el documento?',
'Procesador de textos',MB_YESNOCANCEL+MB_ICONINFORMATION) **of**

ID_YES: //en caso de que devuelva si, Guardar el documento, y despues abrir el document que quiera el usuario

begin

guardar1.Click;

if OpenFileDialog1.Execute **then**

Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

end;

ID_NO: //en caso de que devuelva no, abrir el documento sin guardar el otro

begin

if OpenFileDialog1.Execute **then**

Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

end;

ID_Cancel: //no hacer nada si devuelve Cancelar

end

else //En caso de que no este modificado abrir el documento nuevo

if OpenFileDialog1.Execute **then**

Memo1.Lines.LoadFromFile(OpenDialog1.FileName);

end;

En el elemento **Salir** pondríamos lo siguiente:

begin

If Memo1.Modified **then**

Case Application.MessageBox('¿Desea guardar el documento?', 'Procesador de textos', MB_YESNOCANCEL+MB_ICONINFORMATION) **of**

ID_YES: //En caso de que el usuario pulse si Guardar y después Cerrar

begin

guardar1.Click;

Application.Terminate;

end;

ID_NO: Application.Terminate; //En caso de que el usuario pulse no Cerrar

ID_Cancel:

end

else

Application.terminate; //En caso de que no se modifique Cerrar

end;

Ahora en el evento OnCloseQuery (Pedir cerrar aplicación) el formulario 1(form1) debemos poner el siguiente código, mas o menos el mismo del elemento **salir**:

```
begin  
If Memo1.Modified then  
    Case Application.MessageBox('¿Desea guardar el  
documento?', 'Procesador de  
textos', MB_YESNOCANCEL+MB_ICONINFORMATION) of  
        ID_YES:  
            begin  
                guardar1.Click;  
                Application.Terminate;  
            end;  
        ID_NO: Application.Terminate;  
        ID_Cancel: CanClose:=False; //No permitir que se cierre  
    end  
    else  
        Application.terminate;  
    end;  
end.
```